VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Výukový program pro kompresi dat

Educational System for Data Compression

2012 **Bc. Markéta Migdalová**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Markéta Migdalová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Výukový nástroj pro kompresi dat
Educational System for Data Compression

Zásady pro vypracování:

Cílem práce bude vytvořit systém pro podporu výuky v předmětu Komprese dat. Systém bude obsahovat základní rozhraní a definici pluginů pro rozšíření systému. Systém bude obsahovat základní pluginy pro výuku a testování základních kompresních algoritmů a technik potřebných pro kompresi dat - entropie, Shannon-Fanovo kódování, Huffmanovo kódování, aritmetické kódování.

Systém bude obsahovat:
1. Analýza požadavků a podrobná specifikace zadání dle náplně předmětu Komprese dat.
2. Návrh systému a definice rozhraní
3. Implementace systému
4. Implementace pluginů
5. Testování systému.

Seznam doporučené odborné literatury:

Data Compression: The Complete Reference, David Salomon, 4. ed., Springer, 2007

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

Dne: 30.dubna 2012

.....

Podpis

Poděkování

Ráda bych poděkovala Ing. Janu Platošovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Cílem při tvorbě této diplomové práce bylo vytvořit systém pro podporu výuky v předmětu Komprese dat. Vytvořený systém obsahuje základní rozhraní a definici pluginů pro případné další rozšíření systému. Systém obsahuje základní pluginy pro výuku a testování základních kompresních algoritmů a technik potřebných pro kompresi dat - entropie, Shannon-Fanova kódování, Huffmanovo kódování, aritmetické kódování, také metody LZ77, RLE a MTF.

Klíčová slova

Redundance, entropie, dekomprese, ztrátová komprese, bezztrátová komprese, statistické metody, slovníkové metody, LZ77, Huffmanovo kódovaní, Aritmetické kódovaní, jednoprůchodová kompresní metoda, dvouprůchodová kompresní metoda, statické slovníkové metody, semiadaptivní slovníkové metody

Abstract

The goal of this dimploma thesis is to develop a system to support teaching of the subject of data compression. The created system contains the basic definition of the interface and plug-ins for future system expansion. The system contains the basic plug-ins for teaching and testing basic compression algorithms and techniques needed for data compression - entropy, Shannon-Fano coding, Huffman coding, arithmetic coding, the methods LZ77, RLE and MTF.

Key words

Redundancy, Entropy, decompression, loss compression, lossless compression, statistical methods, dictionary methods, LZ77, Huffman coding, arithmetic coding, single-pass compression method, two pass compression methods, static methods dictionary, semi adaptive dictionary methods

Seznam použitých symbolů

Symbol	Jednotky	Význam symbolu
E_i	Bity	Entropie

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
MTF	Move To Front transform	Transformace "Přesuň na začátek"
RLE	Run-length encoding	Kódování délek
LZ77	Data compression algorithm Lempel-Ziv 77	Metoda komprese dat Lempel-Ziv 77
LZ78	Data compression algorithm Lempel-Ziv 78	Metoda komprese dat Lempel-Ziv 78
LZW	Data compression algorithm Lempel–Ziv–Welch	Metoda komprese dat Lempel-Ziv-Welch

Obsah

1 Úvod

Člověk už od pradávna shromažďuje informace, k jejichž uložení potřebuje určitý paměťový prostor. Během času začal používat pro ukládání informací různorodá úložiště, knihy, poznámky, kartotéky a také úložiště elektronická – média. Tato média mají omezenou velikost, a člověk většinou příliš omezené finanční zdroje, na to aby si mohl koupit nepřeberné množství medií.

Při ukládání nebo transportu dat může být problémem jejich objem (datová velikost), například při přenosu informací přes síť s omezenou rychlostí nebo propustností (GSM) nebo při jejich ukládání na nějaké datové úložiště (například kvůli archivaci), ať už na síti, na pevný disk nebo na přenosný disk. Proto se začal řešit problém zmenšení (komprese) velikosti datových souborů, (např. komprimace hovorů mobilním telefonem). Vzniklo mnoho kompresních algoritmů, využívající různé způsoby kódování, se snahou odstranit ze souboru nadbytečné (redundantní) informace a zvýšit entropii dat.

Důležité pojmy:

- *Kompresor (Kodér, Enkodér)* - Zařízení, člověk nebo počítačový program, který převádí data z jednoho formátu do jiného. Vstupem kompresoru jsou původní data a výstupem data komprimovaná.
- *Dekompresor (Dekodér)* - Zařízení, člověk nebo počítačový program, který dělá reverzní práci ke kompresoru. To znamená, že jeho vstupem jsou komprimovaná data a na výstupu jsou data dekomprimovaná.
- *Zdrojová data* - Původní data před samotnou kompresí.
- *Komprimovaná data* - Data na výstupu kompresoru.
- *Dekomprimovaná data* - Dekomprimovaná data jsou data na výstupu dekompresoru. Je snaha o získání dat původních, ale ne vždy je to možné nebo nutné.

2 Rozdělení kompresních metod

Kompresní metody se dělí podle míry ztrátovosti informací během komprese a dekomprese. Dále podle cílového formátu dat, některé kompresní metody se používají například jen pro kompresi obrazu (JPEG) nebo zvuku (MPEG). Mimo jiné je můžeme dělit podle kompresního a dekompresního algoritmu a to na **Symetrické –** algoritmus komprese i dekomprese je stejný, pouze při dekompresi se provádí jakoby ”pozpátku” a na **asymetrické –** kompresní a dekompresní algoritmus se liší. Rozlišujeme také **jednoprůchodové kompresní metody** a **více-průchodové kompresní metody.**

2.1 Rozdělení kompresních metod dle ztrátovosti

Díky snaze odstranit redundanci v co největší míře se často stává, že dojde ke ztrátě některých, více či méně důležitých informací, takovéto algoritmy komprese dat nazýváme **ztrátovou kompresí**.

2.1.1 Ztrátová komprese

Při použití těchto ztrátových algoritmů nejsme schopni zpětně zcela správně zrekonstruovat (dekomprese) původně komprimované informace (soubory), informace jsou během komprese pozměněny nebo úplně ztraceny (například méně významné informace jsou zcela vypuštěny). U některých typů dat je také možné si následně chybějící informaci domyslet - například u komprese a dekomprese obrazu a zvuku - není třeba vždy přenášet všechny informace, někdy je také možnost soubory zjednodušit a předpokládat zanedbatelnost ztracených informací. Ztrátovou kompresi nazýváme také jako *nepřesnou* nebo *nevratnou kompresi*.

U některých typů dat není vhodné, aby ke ztrátě informací docházelo, v tom případě musíme využít tzv. **bezeztrátovou kompresi**. U níž sice ke ztrátě informace nedochází, bohužel tomu tak ale je na úkor rychlosti samotné komprese a její účinnosti.

2.1.2 Bezeztrátová komprese

Jinak také *přesná* nebo *vratná komprese*. U tohoto typu komprese nedochází ke ztrátě nebo zkreslení informací a při dekompresi jsme schopni zpětně získat původní data, v podobě v jaké byla před použitím komprese, nicméně takováto komprese bývá většinou podstatně méně účinná, a komprimovaný soubor není výrazně menší než soubor původní.

Bezeztrátovou kompresi dále dělíme na algoritmy tvořící **slovníky,** na jejichž základě poté dochází k samotné kompresi a na algoritmy založené na **statistických** údajích získaných z dat.

2.2 Rozdělení kompresních metod dle principu

Mezi základní dělení kompresních metod patří také rozdělení na metody využívající slovníky a metody pracující na základě statistik zdrojových dat.

2.2.1 Slovníkové metody komprese

Slovníkové metody komprese procházejí data a snaží se najít opakující se data, u nichž poté využívají odkazů na dříve zkomprimovaná data. To znamená, že do zkomprimovaného souboru se uloží jen první výskyty dat (znaku, sekvence znaků apod.). Čím je častější výskyt jednotlivých dat/znaků v komprimovaném souboru, tím úspěšnější bude samotná komprimace.

Mezi slovníkové metody komprese patří například metody LZ77, LZ78, LZW84, LZMA. Tato diplomová práce se bude dále zabývat pouze metodou LZ77, nicméně by bylo vhodné později aplikaci rozšířit o další výukové celky.

2.2.2 Statistické metody komprese

Statistické metody komprese nejdříve počítají četnosti výskytů jednotlivých znaků či jejich skupin a na jejich základě se snaží přiřadit jim určité označení, nejčastěji malý počet bitů.

Patří mezi ně například Huffmanovo kódování, Shannon-fanovo kódování a Aritmetické kódovaní.

2.3 Rozdělení kompresních metod podle počtu průchodů

Metody komprese se dále dělí dle počtu průchodů kompresního algoritmu zdrojovými daty. Některé metody komprimují zdrojová data hned při prvním a jediném průchodu a nepotřebují k tomu žádné informace, které by bylo nutné vytvořit v dřívějším průchodu. Jiné metody naopak využívají informací a zdrojů (například slovníky, statistiky), které byly vytvořeny či vypočteny v předchozím průchodu zdrojovými daty.

2.3.1 Dvouprůchodové kompresní metody

Dvouprůchodové kompresní metody procházejí zdrojová data ve dvou fázích. V první fázi se například počítají četnosti výskytů jednotlivých symbolů. Druhá fáze komprese dat poté využívá výsledků výpočtů první fáze kompresního algoritmu.

Mezi dvouprůchodové kompresní metody patří například Shannon-fano kódování.

2.3.2 Jednoprůchodové kompresní metody

V případě jednoprůchodových kompresních metod probíhá komprese v jednom jediném průchodu zdrojovými daty.

Jednoprůchodová kompresní metoda je například metoda LZ78.

2.4 Entropie

Při ukládání dat se většinou klade důraz na jejich snadné využití, na jejich přístupnost a přehlednost ale nehledí se na vznikající redundance. A když někde vznikají redundance, vzniká taky zbytečně využitý prostor, proto všechny kompresní metody využívají metody k odstranění redundancí.

2.4.1 Druhy redundancí

- opakování symbolů
- opakování slov či frází z těchto symbolů složených
- kontextové závislosti (např. v angličtině následuje za *Q* většinou *U*)
- neefektivní přímá reprezentace
- nejednotným rozložením symbolů v textu

Při kompresi dat je pro nás entropie důležitá proto, že nám pomáhá určit, odpovídá-li velikost dat množství informací v datech obsažených. To znamená, jestli je vhodné a možné odstranit opakující se výskyty dat a zjištěné redundance a tím snížit datový prostor, který je potřebný k jejich uložení. Dnes nejsme schopni zjistit přesné množství informace obsažené v datech, proto se využívá entropie.

Její zápis uvedený v následujících odstavcích je přizpůsoben tak, aby entropie byla vyjádřena v bitech, neboť ty jsou základní jednotkou informace používanou při kódovaní dat.

Definice entropie: Nechť data se skládají z *n* různých prvků

$$a_1, a_2, a_3, \ldots, a_n$$

a tyto prvky se v datech vyskytují s pravděpodobnostmi

$$p_1, p_2, p_3, \ldots, p_n$$

Pak množství informace, která je reprezentována prvkem a_i , udává jeho entropie

$$E_i = -\log(p_i) \tag{2.1}$$

Entropie E_i vyjadřuje, jak velkou informaci nese výskyt prvku a_i. Je zřejmé, že čím je pravděpodobnost výskytu prvku a_i menší, tím je jeho entropie a tedy i míra informace větší. (1)

2.5 Kompresní poměr

Kompresní poměr je hodnota, pomocí níž se snažíme vyjádřit efektivitu provedené komprimace. Kompresní poměr určíme jako poměr délky výstupního souboru k délce vstupního soboru. Snahou je dosáhnout kompresního poměru menšího než 1 a co nejvíce se blížícího 0, pokud by tedy kompresní poměr byl větší než 1, pak nedošlo ke komprimaci souboru nýbrž k jeho zvětšení (expanzi).

Kompresní poměr vyjádřený v procentech:

$$kompresní\ poměr = \frac{délka\ výstupního\ souboru}{délka\ vstupního\ souboru} * 100 \tag{2.2}$$

Mezi další sledované údaje o kompresi dat patří také rychlost komprese a dekomprese dat. Přičemž víc nám většinou závisí na rychlosti komprimace, jelikož bývá většinou pomalejší (1).

3 Statistické kompresní metody

Statistické kompresní metody počítají četnosti výskytů jednotlivých symbolů a podle nich poté pravděpodobnosti jejich výskytů. Podle pravděpodobností výskytů jednotlivých znaků potom dále soubor komprimují dle pravidel dílčích kompresních algoritmů. To znamená, že soubor se prochází minimálně dvakrát - jednou aby se spočítaly pravděpodobnosti výskytů znaků, a podruhé za účelem samotné komprimace. Tento způsob komprese dat se hromadně označuje jako **dvouprůchodová kompresní metoda.** Nejznámějšími dvouprůchodovými kompresními metodami jsou:

- Huffmanovo kódovaní
- Shannon-fanovo kódovaní
- Aritmetické kódovaní

3.1 Huffmanovo kódovaní

Huffmanovo kódování je statistická víceprůchodová kompresní metoda, která je běžně používaná několika populárními programy osobních počítačů.

Složitost:

- Časová složitost algoritmu: O (n + | Σ | * log | Σ |)
- Prostorová složitost algoritmu: O (| Σ |) (1)

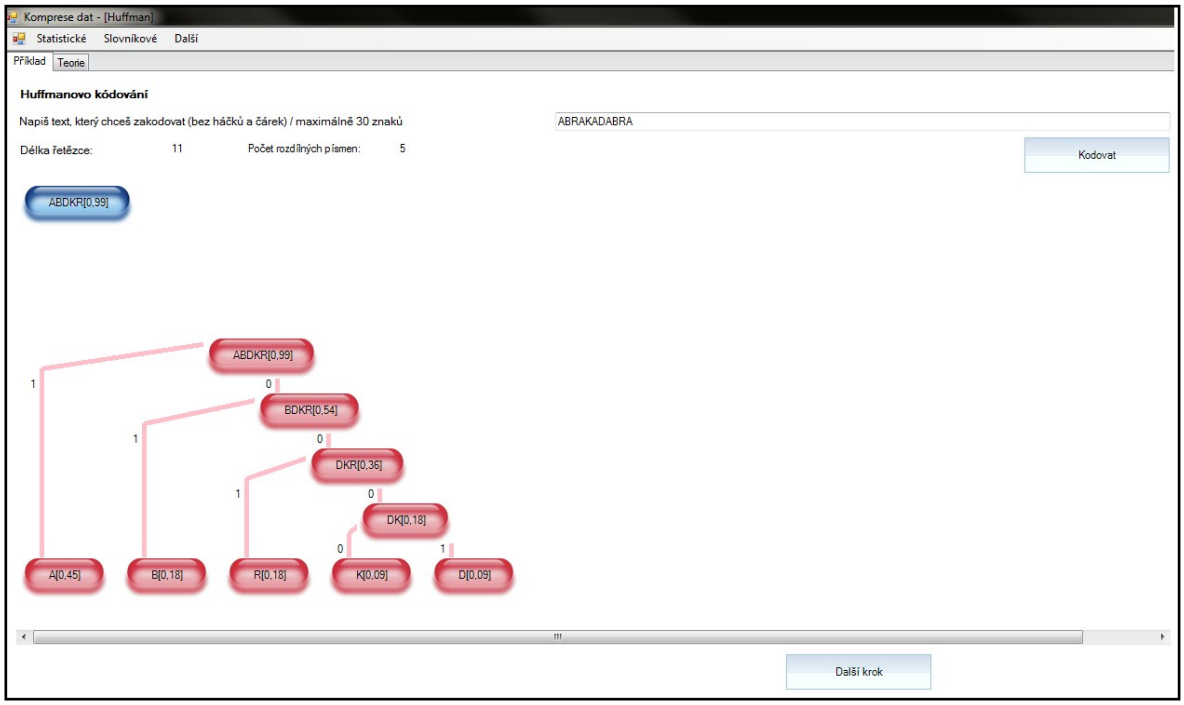
Huffmanovo kódovaní je známo již od roku 1951, kdy student elektrotechniky David Huffman na Ohio State University a jeho spolužáci v předmětu Teoretická Informatika dostali na výběr, zda složí běžnou zkoušku na konci roku nebo vypracují seminární práci. Jako téma seminární práce jim profesor Robert M. Fano (jeden z tvůrců shannon-fanova kódování) zadal nalezení nejkratšího možného dekódovatelného prefixového kódu. V té době ještě nebyl tento problém účinně vyřešen, což ovšem studenti nevěděli. David Huffman algoritmus vyřešil, a to velmi elegantním a prostým způsobem pomocí binárního stromu, který byl velmi užitečný v praxi. David Huffman nikdy nepožádal o patent na tento svůj objev, protože jak vždy říkal svému synovi: Je to jen algoritmus.

Dnes je Huffmanovo kódování v různých variantách (např. adaptivní) používáno v některých komprimačních algoritmech např.: JPEG, HuffYuv, MP3, BZIP2, PKZIP. Například open-source program Bzip2 zpočátku používal aritmetické kódování, ale protože firma IBM posléze získala patenty na aritmetické kódování, rozhodli se použít Huffmanova kódovaní.

Princip binárního Huffmanova kódování

V prvním průchodu souborem spočítá algoritmus pravděpodobnosti výskytů všech jednotlivých znaků vstupní abecedy. Metoda vytvoří listy binárního stromu, které jsou tvořeny jednotlivými znaky seřazenými sestupně na základě pravděpodobností výskytů.

V dalším průchodu na základě těchto pravděpodobností vytváří binární strom, kdy při každém větvení na dvě samostatné větve dostane jedna větev ohodnocení 0 a druhá ohodnocení 1. V každém kroku se poté sečte pravděpodobnost dvou nejníže postavených symbolů, a jejich symbolem se v danou chvíli stává zřetězení obou symbolů uzlů předcházejících. Přičemž platí, že znaky s největší pravděpodobností výskytu jsou nejblíže samotnému kořeni stromu. Takto je každému znaku přiřazen binární kód, který vznikne zřetězením hodnot, jimiž jsou ohodnoceny hrany od kořene k listu (znaku) (2). Viz obr. 3.1.



Obrázek 3.1: Huffmanovo kódování

To znamená, že například písmeno B má prefix “01” a písmeno D má prefix “0001”.

Samotný výstup kompresního algoritmu je až nahrazení vstupních znaků prefixovými kódy.

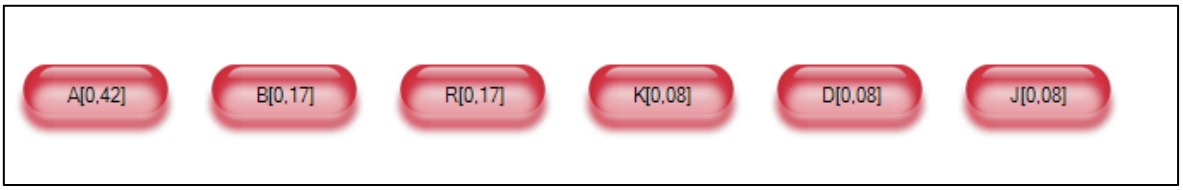
Příklad

Mějme příklad z obrázku 3.1.

Znak	*A*	*B*	*R*	*J*	*D*	*K*
Počet Výskytů	*5*	*2*	*2*	*1*	*1*	*1*
Pravděpodobnost	*0,42*	*0,17*	*0,17*	*0,08*	*0,08*	*0,08*

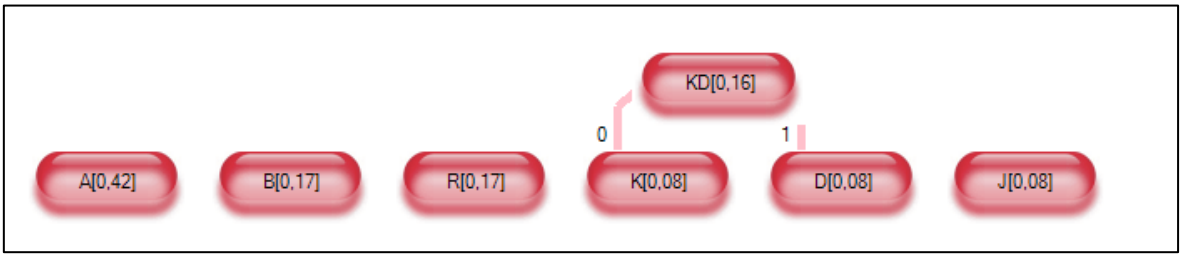
Obrázek 3.2: Huffmanovo kódování

V prvním kroku vykreslíme všechny listy stromu.



Obrázek 3.3: Huffmanovo kódování

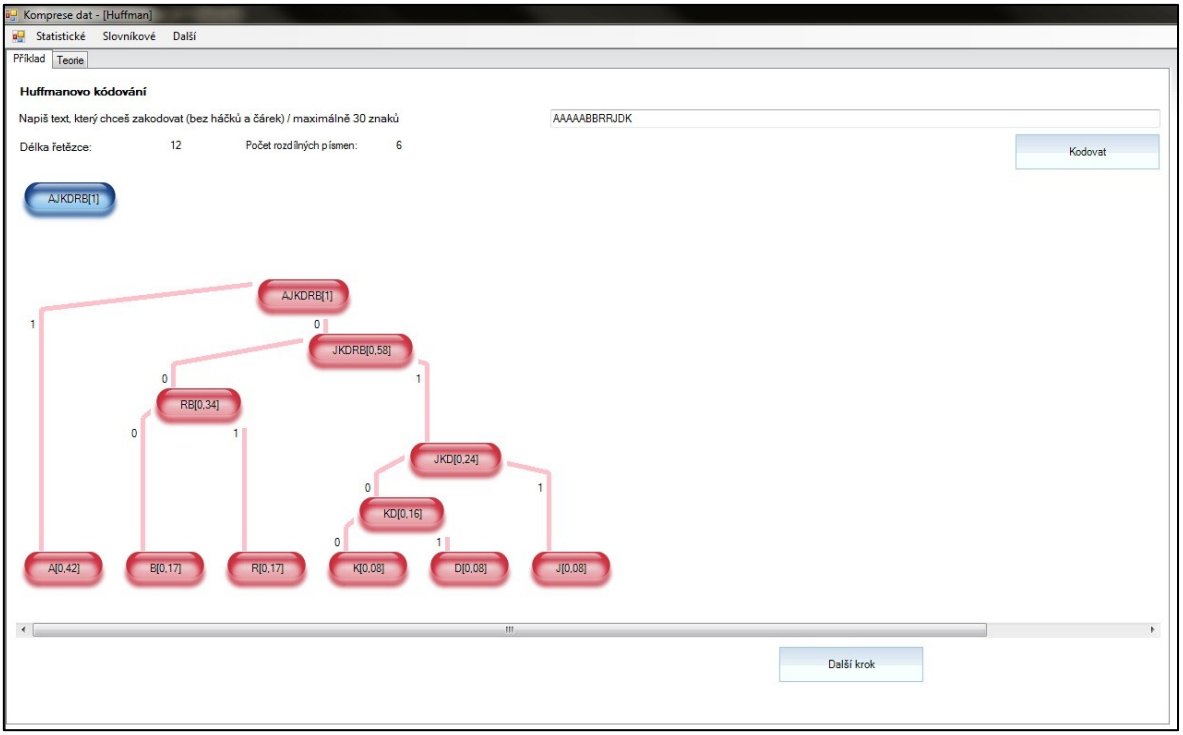
Nyní spojíme dva listy stromu s nejmenší četností "K" a "D" a vytvoříme nový uzel "KD" s pravděpodobností 0,16, první vytvořený spoj ohodnotíme hodnotou „0" a druhý "1".



Obrázek 3.4: Huffmanovo kódování

V dalším kroku spojíme opět dva uzly s nejmenší pravděpodobností, což jsou uzly "KD" a „J" do nového uzlu s pravděpodobností 0,24.

Takto pokračujme tak dlouho, než nám zbude jeden jediný uzel, a to kořen se znaky „AJKDRB" a hodnotou 1.



Obrázek 3.5: Huffmanovo kódování

Princip dekomprese Huffmanova kódu

Dekomprese je při znalosti binárního stromu, vytvořeného při kompresi velmi snadná. Spočívá jen v nahrazení prefixových kódů původními znaky.

Varianty Huffmanova kódovaní

Dynamická (adaptivní) verze – tato varianta reaguje na změnu pravděpodobností výskytu vstupních znaků. Tato verze je samozřejmě pomalejší, ale zato umožňuje dosáhnout lepšího kompresního poměru, jelikož odpadá nutnost uchovávat binární strom. Dynamická verze se řadí mezi jednoprůchodové kompresní algoritmy.

Při užití Dynamického Huffmanova kódovaní předpokládáme, že veškeré vstupní znaky mají četnost výskytu nejméně 1, díky čemuž nepotřebujeme soubor procházet pro výpočet četností znaků. Dynamického Huffmanova kódování existuje také několik variant, např. Vitterův algoritmus, FGK kódování.

Oba dva uvedené algoritmy využívají tzv. sourozeneckou vlastnost, k vysvětlení použiji definici jednoho z tvůrců FGK kódování : (Gallager, 1978, s. 669) „Binární strom má sourozeneckou vlastnost, v případě, že každý uzel (kromě kořene) má sourozence a v případě, že uzly mohou být seřazeny s nerostoucí pravděpodobností (výskytu) tak, že má každý uzel v posloupnosti za souseda svého sourozence.

N-ární Huffmanovo kódovaní – Patří mezi ně i binární Huffmanovo kódování popsané výše. Princip spočívá v použití {0,1,…,N-1} znaků k zakódování vstupního řetězce. U binárního huffmanova kódovaní se spojují vždy jen dva znaky s nejnižší pravděpodobností, u obecně *n*-árního kódování se spojí *n* znaků s nejnižší pravděpodobností výskytu. Potom vzniká n-ární strom. Protože u některých *n* můžou vznikat problémy s vytvářením stromu (například je – li znaků méně než hodnota *n*), přidávají se pomocné znaky s pravděpodobností výskytu 0.

N-ární huffmanovo kódování by mohlo být předmětem dalšího rozšíření výukové aplikace.

Kanonické Huffmanovo kódovaní (Huffman-Shannon-Fano)- Odstraňuje nutnost uchovávání binárního stromu. Princip spočívá v pevném přiřazení binárního kódu znakům, podle předem určených pravidel. K dekompresi poté stačí znát pravidla, která byla při kompresi dodržována. Této technice se někdy říká Huffmanovo-Shannon-Fanovo kódování, protože je úspěšné jako huffmanovo kódování ale využívá váhy pravděpodobností jako Shannon-Fanovo.

Kanonické huffmanovo kódování by mohlo být předmětem dalšího rozšíření výukové aplikace.

Zhodnocení

Výhodami Huffmanova kódování jsou velmi rychlá komprese i dekomprese a nepříliš velké nároky na operační paměť.

Mezi nevýhody patří nutnost uchovávání binární stromu pro případ pozdější dekomprese a fakt, že algoritmus si nevšímá opakování řetězců (na rozdíl od slovníkových metod), to znamená, že má slabý kompresní poměr.

Programy a multimediální formáty využívající Huffmanova kódování

Bzip2, WinZip, WinRar, Jpeg, MP3, JPEG, HuffYuv, PKZIP

3.2 Shannon-Fano kódování

Shannon-fano kódování je statistická víceprůchodová kompresní metoda. Metoda je velmi podobná metodě Huffmanova kódování, ale je méně efektivní, a proto nebývá běžně používaná pro kompresi dat v běžně rozšířených programech osobních počítačů.

Metoda podobně jako předchozí metoda používá ke kódování binární strom, který se však liší v základní konstrukce, a to tím, že binární strom Huffmanova kódování se vykresluje od listů směrem ke kořeni, kdežto strom Shannon-fano kódování se vykresluje od kořene směrem k listům binárního stromu.

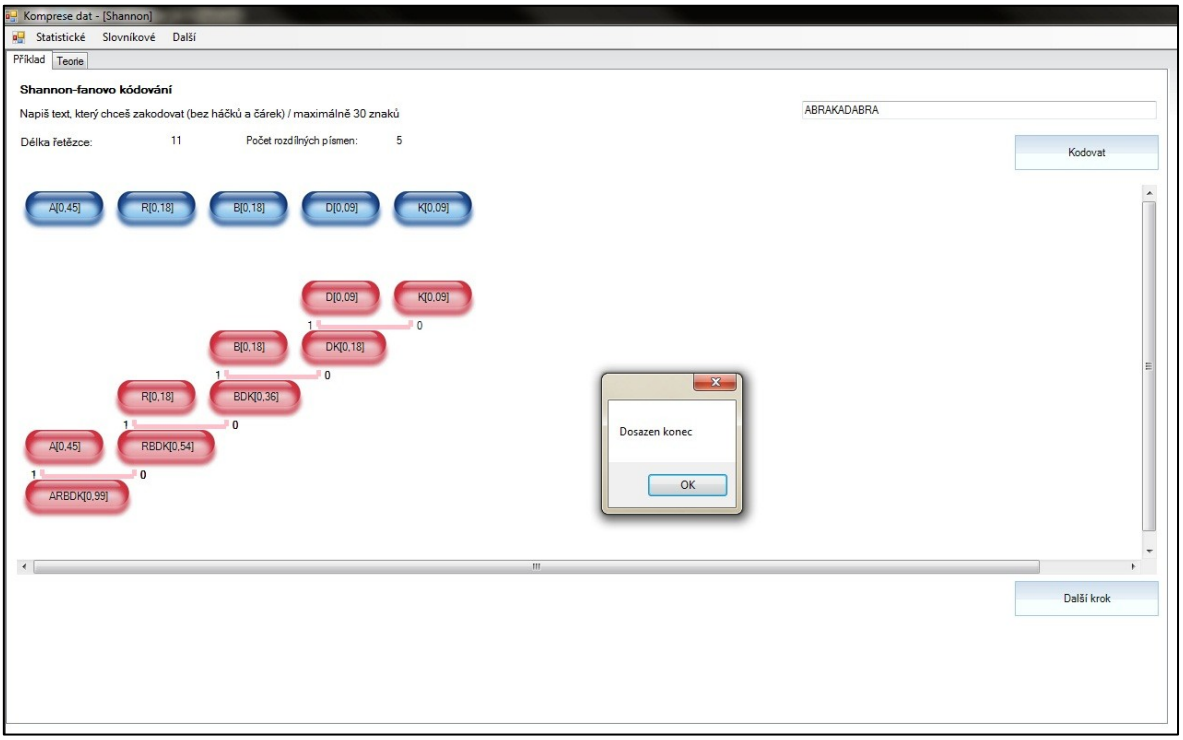
Složitost:

- Časová složitost algoritmu**:** O (n + | Σ | * log | Σ |)
- Prostorová složitost algoritmu: O (| Σ |) (1)

Shannon-fanovo kódování je známo již od roku 1949. Jako vynálezci jsou uváděni Americký matematik, elektrotechnik a zakladatel teorie informace Claude Elwood Shannon, americký profesor elektrotechniky a informatiky Robert Mario Fano, původně narozený v Itálii.

Princip binárního Shannon-fanova kódování

V prvním průchodu vstupním řetězcem se spočítají četnosti výskytů jednotlivých znaků. Kořenem binárního stromu je řetězec složený ze všech znaků, které jsou součástí kódované zprávy. Další kroky a větvení se provádí vždy tak, že se rekurzivně množina znaků dělí na dvě přibližně stejně velké podmnožiny podle četnosti výskytů znaků ve vstupním řetězci. Každé takto vzniklé podmnožině se přidělí hodnota 0 nebo 1. Binární krok příslušející jednotlivým znakům se potom vyčte jako posloupnost znaků 0, 1 při cestě od kořene k listu. Většinou se levým větvím binárního stromu přiřazuje hodnota 1 a pravým větvím se přiřazuje 0(2).



Obrázek 3.6: Shannon-fanovo kódování

V tomto případě bude například slovo K zakódováno jako „0000“ a znak R jako “01”.

Příklad

Znak	*A*	*R*	*B*	*J*	*D*	*K*
Počet Výskytů	*5*	*2*	*2*	*1*	*1*	*1*
Pravděpodobnost	*0,42*	*0,17*	*0,17*	*0,08*	*0,08*	*0,08*

Tabulka 3.7: Shannon-fanovo kódování

V prvním kroku vykreslíme kořen stromu.

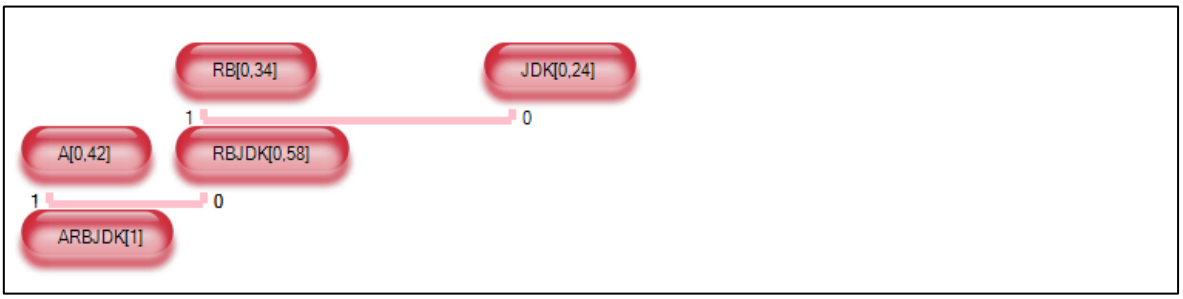
Obrázek 3.8: Shannon-fanovo kódování

Nyní se snažíme rozdělit pole znaků (seřazené podle pravděpodobnosti) na dvě přibližně stejné poloviny (podle součtů pravděpodobností).

V tomto případě to bude znak “A” s pravděpodobností 0,42 a znaky “RBDK” s pravděpodobností 0,58.

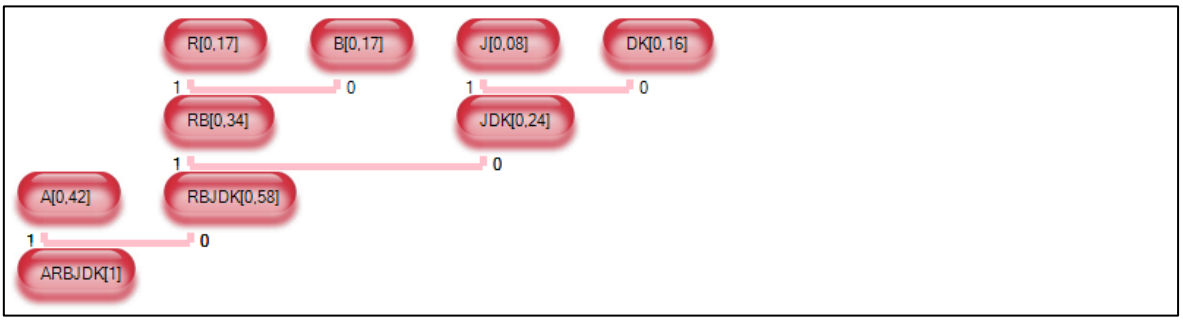
Obrázek 3.9: Shannon-fanovo kódování

Jelikož uzel "A" je listem binárního stromu, rozdělujeme nyní uzel "RBJDK" na dvě přibližně stejné poloviny. A to na "RB" s pravděpodobností 0,34 a na "JDK" s pravděpodobností 0,24.



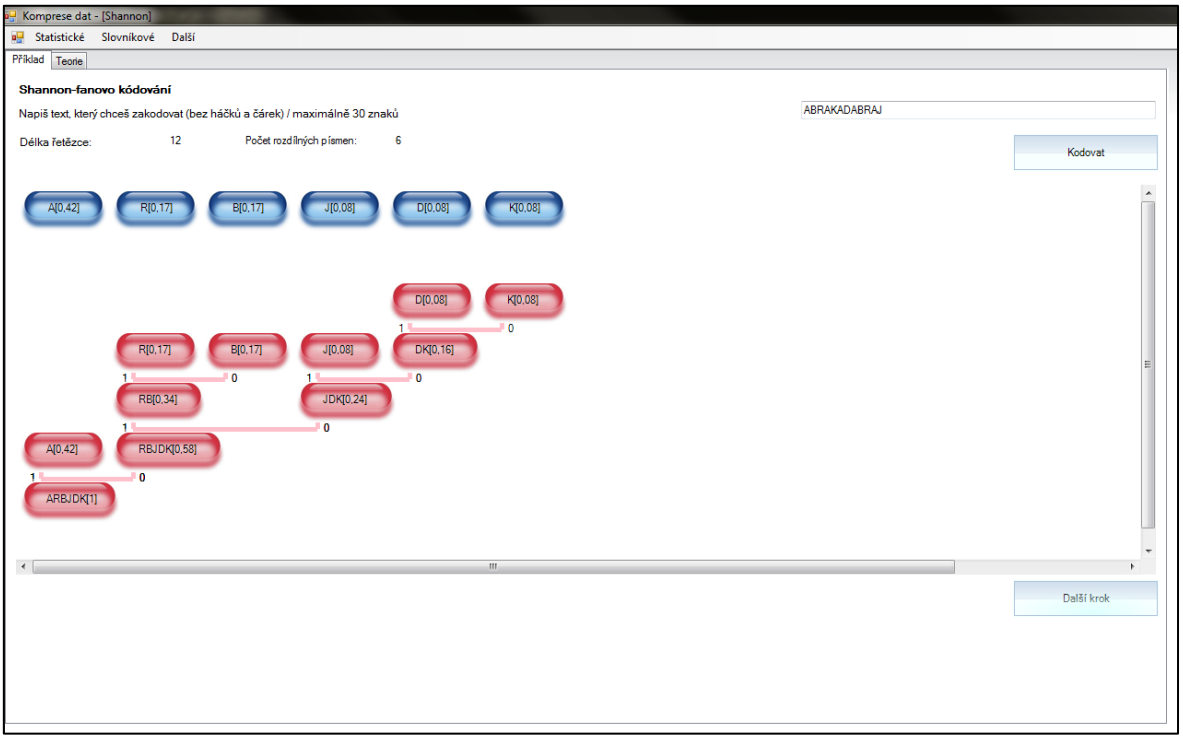
Obrázek 3.10: Shannon-fanovo kódování

Nyní dělíme uzel "RB" a na listy „R" s pravděpodobností 0,17 a "B" s pravděpodobností 0,17. A uzel "JDK" na list "J" s pravděpodobností 0,08 a uzel "DK" s pravděpodobností 0,16.



Obrázek 3.11: Shannon-fanovo kódování

Poté nám zbývá pouze rozdělit uzel "DK" na listy "D" a "K" s pravděpodobnostmi 0,08.



Obrázek 3.12: Shannon-fanovo kódování

Princip dekomprese Shannon-fanova kódu

Dekomprese je při znalosti binárního stromu, vytvořeného při kompresi velmi snadná. Spočívá jen v nahrazení prefixových kódů původními znaky.

Srovnání Shannon-fanova kódování a Huffmanova kódování

Výsledná komprese je ve srovnání s Huffmanovým kódováním horší.

Programy využívající Shannon-fanova kódovaní

- WinZip

3.3 Aritmetické kódování

Aritmetické kódování je bezeztrátová statistická víceprůchodová kompresní metoda.

Složitosti:

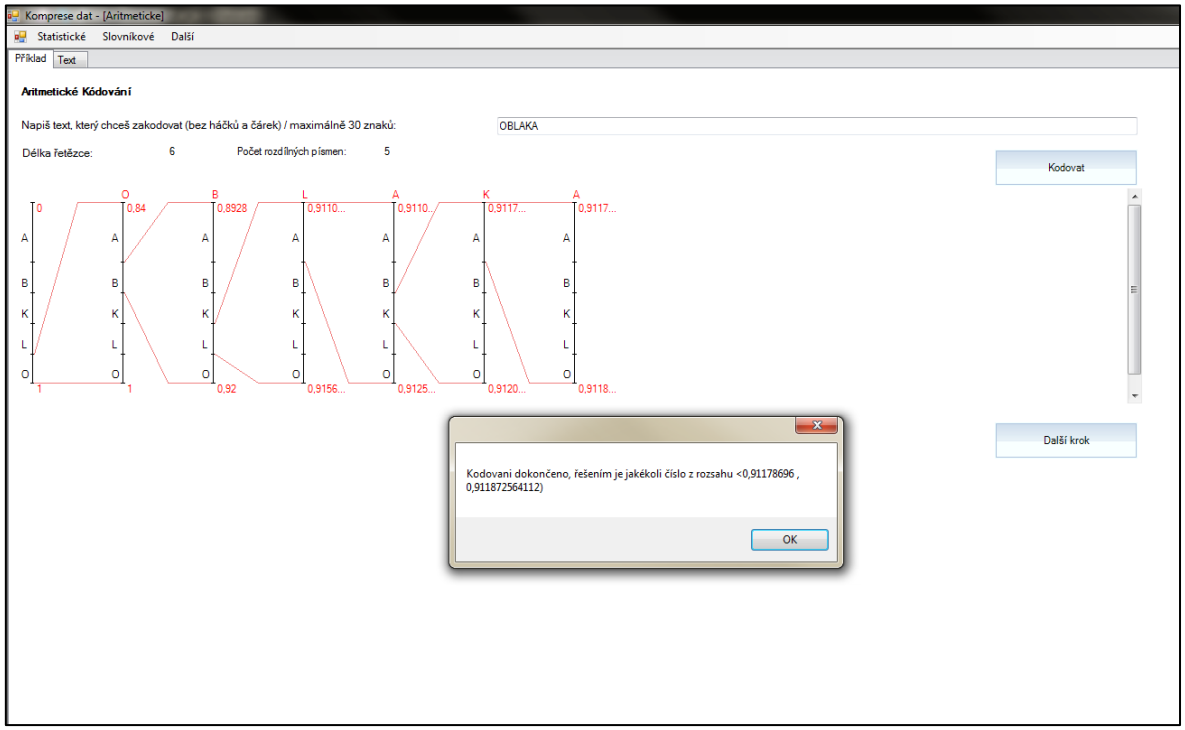
- Časová složitost algoritmu: O (| Σ | + n)
- Prostorová složitost algoritmu: O (| Σ |) (1)

S myšlenkou nahradit komprimovaný řetězec jedním jediným prefixem s minimální popisnou délkou si zahrávali již tvůrci Huffmanova a Shannon-fanova kódování, ale poprvé ji navrhl Peter Elias někdy před rokem 1963 v nikdy nepublikované práci. Tato verze aritmetického kódování však s sebou

nesla vysoké nároky na vyrovnávací paměť. S vylepšením přišli Jorma J. Rissanen (*20. 8. 1932 Finsko) a Richard Pasco v roce 1975 ve své práci.

Princip Aritmetického kódování

Základní myšlenkou Aritmetického kódování je nahrazení sekvence vstupních symbolů jediným číslem z podintervalu intervalu <0,1). V první fázi se stejně jako u předchozích metod vypočítají četnosti výskytů vstupních znaků. V dalším kroku se každému znaku přiřadí příslušná část (podle pravděpodobnosti výskytu) z intervalu <0,1).



Obrázek 3.13: Aritmetické kódování

Samotné kódování vstupních znaků spočívá ve vyhrazení intervalu úměrného pravděpodobnosti výskytu znaku z intervalu vyhrazeného v minulém kroku.

Příklad:

V příkladu na obrázku je pravděpodobnost výskytu jednotlivých znaků a rozdělení intervalů:

Znaka	*Pravděpodobnost P(a)*	*Interval od L(a)*	*Interval do H(a)*
A	*0,3333*	*0*	*0,3333*
B	*0,1667*	*0,3333*	*0,5*
K	*0,1667*	*0,5*	*0,6667*
L	*0,1667*	*0,6667*	*0,8333*
O	*0,1667*	*0,8333*	*1*

Tabulka 3.14: Aritmetické kódování

Jako první kódovaný znak je písmeno “O” to znamená, že z počátečního interval <0,1) vybereme interval přiřazený písmenu “O”, což je <0,8333;1).

Výpočet intervalu <d; h):

$$L = L + L(a) \cdot (H - L) \tag{3.1}$$

$$L = L + H(a) \cdot (H - L) \tag{3.2}$$

Pro výpočet interval prvního znaku je L=0 a H=1 (2).

Další kódovaný znak je písmeno „B“, jeho interval <L; H) se vypočte pomocí dosazení do vzorce:

Známe:

$$L = 0{,}8333$$

$$H = 1$$

$$L(B) = 0{,}3333$$

$$H(B) = 0{,}5$$

Výpočet:

$$L = 0{,}8333 + 0{,}3333 \cdot (0{,}1667) = 0{,}88886111 \cong 0{,}89$$

$$H = 0{,}8333 + 0{,}5 \cdot (0{,}1667) = 0{,}91715 \cong 0{,}92$$

V tuto chvíli máme zakódován řetězec “OB”jako jakékoli číslo z intervalu <0,89;0,92)

Pro řetězec “OBL” je $L \cong 0{,}9110, H \cong 0{,}9156$

Pro řetězec “OBLA” je $L \cong 0{,}9110$, H $\cong 0{,}9125$

Pro řetězec “OBLAK” je $L \cong 0{,}9117$, H $\cong 0{,}9120$

Pro řetězec “OBLAKA” je $L \cong 0{,}9117$, $H \cong 0{,}9118$

Výsledným prefixem slova OBLAKA je jakékoli číslo z intervalu <0,9117; 0,9118).

Srovnání aritmetického kódování a Huffmanova kódování

Nejlepšího kompresního poměru při využití huffmanova kódu dosáhneme, jsou-li četnosti výskytů jednotlivých znaků mocninami čísla 2. U Aritmetického kódování tento problém odpadá.

Výhody aritmetického kódovaní

Co se týče kompresního poměru, tato komprimační metoda není příliš účinná. Nicméně se využívá jako doplňková metoda k některým adaptivním slovníkovým metodám (např. LZW).

Programy používající aritmetické kódování

- Bzip2
- Jpeg

4 Slovníkové kompresní metody

Slovníkové kompresní metody jsou vhodné zejména pro použití ke kódování textu, zvláště pak, pokud se v textu často opakují stejné sekvence znaků. U těchto metod se využívá opakování posloupností znaků, přičemž poprvé, když se na danou posloupnost narazí, uloží se do slovníku a při dalších výskytech se pouze odkazuje na tento záznam.

Slovníkové metody dělíme **na statické slovníkové metody** – to jsou ty, které používají ke kódování slovník, který je dopředu předpřipraven a na **semiadaptivní slovníkové metody** – slovník je tvořen na základě vstupních dat a **adaptivní slovníkové metody** - slovník je tvořen teprve v průběhu kódování.

4.1 Statické slovníkové metody

Jak již bylo zmíněno výše, statické slovníkové metody při komprimaci používají již předem připravený slovník frází.

Statické slovníkové metody můžeme podle principu vytváření slovníku rozdělit na dva základní typy:

1. Slovník obsahuje fráze určité délky, například dvojice symbolů, které se vedle sebe často vyskytují. Těmto frázím se říká *n-gramy*.
2. Slovník obsahuje fráze nestejné délky. Tyto fráze jsou vybírány podle frekvence jejich běžného výskytu. Takže například slova “nebo”, “ale” nebo třeba slovo “protože”.

Výhodou statických slovníkových metod je, že při znalosti slovníku nepotřebujeme ke komprimovaným datům slovník přikládat a také se nezdržujeme jeho vytvářením při komprimaci.

4.2 Semiadaptivní slovníkové metody

Statické slovníkové metody mají tu nevýhodu, že nejsou připraveny přímo pro konkrétní kódovaná data, což v případě, že kódovaná data budou obsahovat jen zlomek frází ze slovníku, bude mít za následek nepříliš dobrý výsledný kompresní poměr.

Jako řešení se nabízí semiadaptivní slovníkové kompresní metody, u nichž dochází k vytvoření slovníku až podle aktuálně kódovaných dat. Nevýhodou semiadaptivní slovníkové metody je, že se musí společně s komprimovanými daty ukládat také slovník. Další nevýhodou této metody je fakt, že data procházíme dvakrát – jde tedy o víceprůchodovou kompresní metodu. A to poprvé pro vytvoření slovníku frází a podruhé pro samotnou komprimaci.

4.3 Adaptivní slovníkové metody (metody s rostoucím slovníkem)

U tohoto typu metod slovníkové komprese předpokládáme, že se všechny znaky vyskytují se stejnou pravděpodobností. Teprve v průběhu komprimace dat upravujeme slovník. Tento postup

odstraňuje nevýhody obou předchozích typů slovníkových metod. Dochází totiž pouze k jedinému průchodu komprimovaným souborem a navíc není nutné ukládat slovník, jelikož při dekompresi si ho dekomprimační algoritmus opět snadno vytvoří z komprimovaných dat.

Další podstatnou výhodou tohoto typu komprese dat je, že je možno komprimovaná data začít odesílat (například po síti) již před úplným dokončením komprimace. Může tedy nastat situace, kdy na jedné straně odesílatel komprimované zprávy nemá všechna data zkomprimovaná a teprve je pro odeslání chystá, a na straně druhé už příjemce dekomprimuje dříve přijatá data.

Mezi adaptivní slovníkové metody patří například LZ77, LZ78 a LZW (LZW84).

4.4 LZ77

Jak již bylo uvedeno výše, LZ77 se řadí mezi slovníkové jednoprůchodové kompresní metody, které při komprimaci nevytvářejí slovník frází (opakujících se částí textu). Tato metoda byla publikována Jacob Ziv (Izraelský profesor počítačových věd, vystudoval v Izraeli a Massachusetts) a Abraham Lempel (Izraelský profesor počítačových věd, původem z Polsko - Ukrajinského města Lwów) v roce 1977 v *A Universal Algorithm for Sequential Data Compression" in the IEEE Transactions on Information Theory (May 1977)*. Odtud také pojmenování komprimační metody: **L**ampel-**Z**iv 19**77**.

4.4.1 Princip metod založených na LZ77

Pro pochopení principu metody si musíme představit posuvné okno, které se postupně přesouvá po kódovaném řetězci od začátku ke konci. Toto okno má pevně stanovenou délku, to znamená počet znaků, které toto okno pojme – aktuálně komprimuje. Metoda v každém kroku hledá opakující se symboly či řetězce, které se již vyskytly ve výstupním řetězci. Výstupem jsou dvě čísla a znak, první číslo určuje pozici, na níž se nachází shoda, druhé číslo délku shody a písmeno určuje první neshodný znak. Při kódovaní prvního znaku zdrojového souboru, nebo znaku, který se ještě v celé dříve zakódované části zdrojového souboru nenachází, je pozice shody 0, délka shody 0 a prvním neshodným znakem je samotný právě komprimovaný znak.

Pro vysvětlení principu této metody použijeme okno o velikosti 4 znaky.

Obrázek 4.1:LZ77

V prvním kroku kompresní algoritmus ještě na výstupu nemá žádné znaky, a proto první znak rovnou přepíše na výstup – na obrázku znázorňuje výstupní soubor řetězec “<0,0,A>“.

Poté se posuvné okno posune o jeden znak a čte znaky “BRAK”, přičemž algoritmus kontroluje, zda se fráze alespoň částečně shoduje s prvním uloženým znakem. Protože ani první znak “B” nyní ještě algoritmus není schopen zakódovat pomocí dříve zakódovaného znaku, výstupem bude “<0,0,A>; <0,0,B>”.

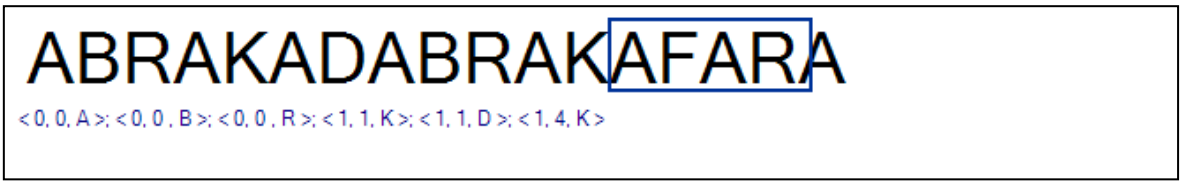
Obrázek 4.2: LZ77

Posuvné okno se posune o jeden znak a nyní čte řetězec “RAKA”. Algoritmus zjišťuje, zda je fráze “RAKA” již obsažena ve výstupním řetězci. První písmeno se opět neshoduje, proto na výstup zapíšeme informaci “<0,0,R>” a posuvné okno se opět přesune o jeden znak.

Nyní čteme řetězec “AKAD”, přičemž se shoduje pouze písmeno „A“. Písmeno “A” má první výskyt na pozici 1, a délka shody je 1, první neshodný znak (znak, který zatím není zakódován) je písmeno “K”. Výstupem tedy v současné chvíli bude “<0,0,A>; <0,0,B>; <0,0,R>; <1,1,K>”.

Posuvné okno se posune o dva znaky, za písmeno K. Čteme nyní řetězec “ADAB”, shoduje se opět pouze písmeno “A”, proto výstupem bude “<0,0,A>; <0,0,B>; <0,0,R>; <1,1,K>; <1,1,D>”.

Posuvné okno se opět posune o dva znaky a nyní čte řetězec “ABRA”, písmeno “A” již v zakódovaném řetězci je, řetězec “AB” také, řetězce “ABR” “ABRA” jsou také již v zakódované části, jelikož algoritmus používá posuvné okno délky 4 nekontroluje už další možné shody a pro výstup použije nejdelší shodu délky 4, od pozice 1 a první neshodný znak je “K”. V této situaci by bylo vhodnější použití posuvného okna délky 8, jelikož shoda by v tu chvíli mohla být délky 6.



Obrázek 4.3: LZ77

Výstupem v tuto chvíli je řetězec “<0,0,A>; <0,0,B>; <0,0,R>; <1,1,K>; <1,1,D>; <1.4,K>” a posuvné okno snímá řetězec “AFAR”, přičemž znak “A” se shoduje s výstupem, první neshodný znak je písmeno “F”, délka shody je 0.

V dalším kroku se shoduje písmeno “A” ale ne sekvence “AR”, proto bude “R” prvním neshodným znakem.

Nyní se čte už pouze znak “A”, jako první neshodný znak bude ukončovací řetězec.

Výstupem je nyní:

“<0,0,A>; <0,0,B>; <0,0,R>; <1,1,K>; <1,1,D>; <1.4,K>; <1,1,F>; <1,1,R>; <1,1,\0>”

4.4.2 **Použití LZ77**

- WinRAR
- 7-Zip
- ZIP

LZ77 bývá používáno kompresními programy jako doplňkový algoritmus k jinému typu kompresní metody. Například kombinace LZ77 a Huffmanova kódování se nazývá DEFLATE a je využíván:

- Gzip
- PNG
- PKZIP

4.4.3 **DEFLATE**

DEFLATE je bezeztrátový kompresní algoritmus založený na kombinaci kompresní metody LZ77 a Huffmanova Kódování., který byl vytvořen za účelem nahrazení jiných výhodných algoritmů, které ale jsou zatížené patentem, a proto není možné je běžně použít. Jako například kompresní metoda LZW.

Metoda DEFLATE by mohla být součástí dalšího rozšíření aplikace pro výuku v předmětu komprese dat.

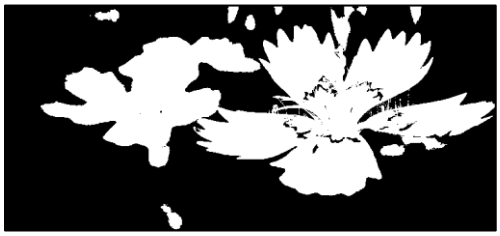
5 Pomocné algoritmy

Při kompresi dat se také používá mnoho pomocných kompresních algoritmů pro předpřipravení dat ke kompresi.

5.1.1 RLE

Jednoduchá bezztrátová kompresní metoda. Česky se název metody překládá jako „přesuň na začátek" nebo "proudové kódování".

Název je odvozen z anglického **R**un-**L**ength **e**ncoding. Metoda komprese RLE využívá faktu, že jsou některá data složena z dlouhých posloupností opakujících se hodnot, jako tomu bývá například u obrázků s nízkým rozlišením.



Obrázek 5.1: obrázek s nízkým rozlišením

Princip RLE

Jde zhuštění opakovaných znaků, které se v souboru vyskytují za sebou. Tato sekvence znaků je zakódována do jednoho paketu RLE, který obsahuje dvě informace:

- Číslo udávající počet znaků sekvence
- Opakující se znak sekvence (4)

Příklad:

Mějme řetězec "bbbbwwwwwwwwwbbbbwwbbbwbbbbb".

Výstupem bude řetězec "4b9w4b2w3b1w5b".

Použití RLE

Přenos dat pro fax – většina faxovaných dokumentů jsou celé bílé jen s občasnými černými body, používá se v kombinaci ještě s jinými druhy kódování (Huffman).

RLE je vhodné používat pro kreslené obrázky.

Zhodnocení

RLE je nevhodné používat, pokud nedochází k častému opakování znaků. Avšak existuje varianta, která pokud délka opakování znaku není vhodná pro použití dvojice (počet opakování, znak) přenáší do výstupního souboru pouze znak.

5.1.2 MTF

Transformace používaná při kompresi dat, která se většinou používá na datech, která byla přichystána BWT. Transformace je reversibilní, při dekompresi nedochází ke zkreslení. Název znamená Move-to-Front, to znamená "Přesuň na začátek". Algoritmus neustále mění pořadí abecedního seznamu znaků, podle jejich posledního výskytu. Abecední seznam přiřazuje indexy všem znakům a to v pořadí 0, 1, 2,…, 255.

První znak bývá zakódován vždy svým vlastním indexem a hned poté je tento znak přesunut na začátek seznamu, kde získává index 0.

Příklad:

Mějme řetězec "BARAKADABRA".

ZNAK	INDEX	ZNAK	INDEX	ZNAK	INDEX
A	0	J	9	S	18
B	1	K	10	T	19
C	2	L	11	U	20
D	3	M	12	V	21
E	4	N	13	W	22
F	5	O	14	X	23
G	6	P	15	Y	24
H	7	Q	16	Z	25
I	8	R	17		

Tabulka 5.1: MTF

B má index 1, zařadíme jej na začátek, před znak "A". Výstup – "1".

ZNAK	INDEX	ZNAK	INDEX	ZNAK	INDEX
B	**0**	**J**	**9**	**S**	**18**
A	**1**	**K**	**10**	**T**	**19**
C	**2**	**L**	**11**	**U**	**20**
…	**...**	**M**	**12**	**V**	**21**

Tabulka 5.2: MTF

V dalším kroku čteme písmeno “A”. Znak “A” přesuneme před znak “B”, získává index 0. Výstup – “1,1”.

Nyní čteme “R” s aktuálním indexem 17, znak ”R” zařadíme před „A“. Nyní má Znak “A” index 1, “B” index 2, „R“ index 0 a “C” index 3. Výstup – “1,1,17”. V tuto chvíli čteme “A”, s indexem 1 a znak se přesouvá na začátek před “R”. Výstup - “0,1,17,1”.

Takto postupujeme až do doby, než zakódujeme poslední znak. V tuto chvíli budeme mít výstup – “1, 1, 17, 1, 11, 1, 5, 1, 4, 4, 2“ a pořadí:

ZNAK	INDEX	ZNAK	INDEX	ZNAK	INDEX
A	0	H	9	S	18
R	1	I	10	T	19
B	2	J	11	U	20
D	3	L	12	V	21
K	4	M	13	W	22
C	5	N	14	X	23
E	6	O	15	Y	24
F	7	P	16	Z	25
G	8	Q	17		

Tabulka 5.3: MTF

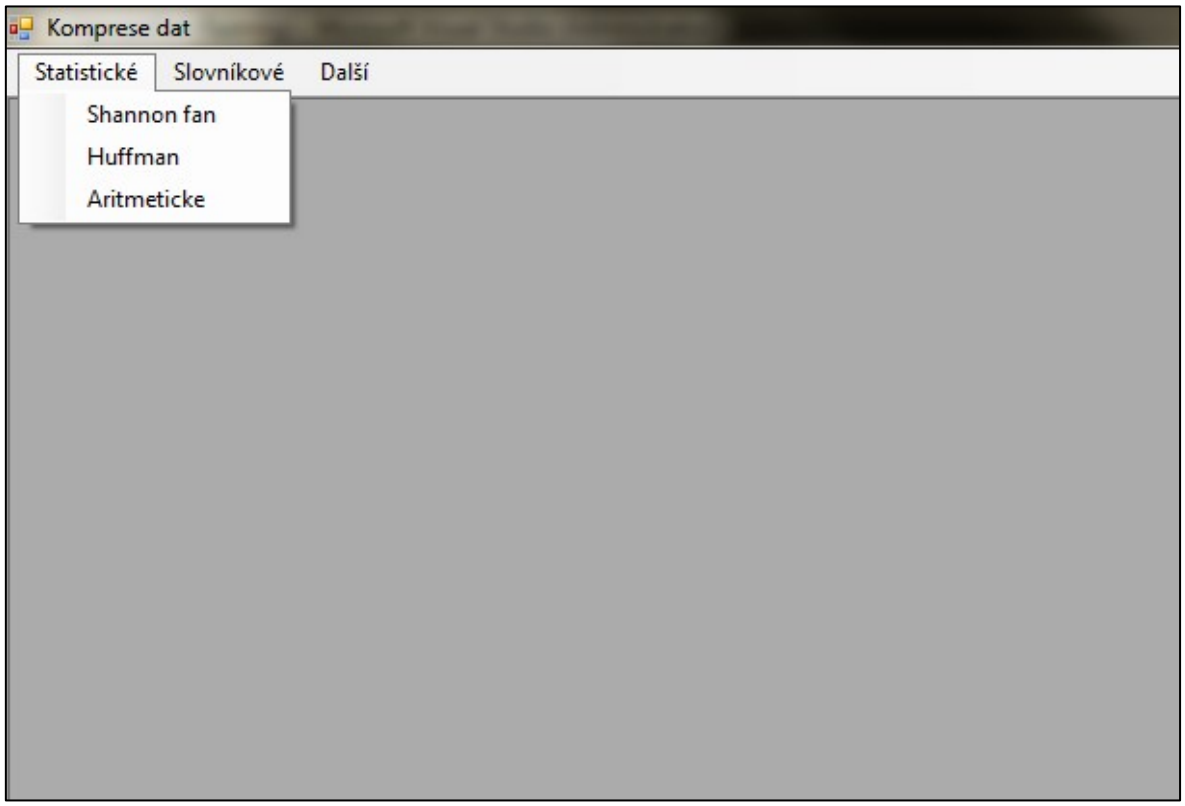
6 Popis implementace

Pro implementaci bylo zvoleno vývojové prostředí Microsoft Visual Studio 2010, jazyk Visual C#.

6.1 Základní orientace

Zadáním bylo vytvořit modulární aplikaci s podporou pluginů. Samotná aplikace obsahuje úvodní část Crossroad, která se načte po spuštění projektu a umožňuje práci s jednotlivými pluginy.

Aplikace obsahuje pluginy Statistics, Dictionary Others, jejichž jednotlivé výukové moduly je možné vybrat v Menu. Formulář má nadefinovánu vlastnost IsMdiContainer=true, přičemž po výběru některé z položek menu se nově spuštěný formulář spustí v rámci MdiContainer.



Obrázek 6.1: Crossroad

Jádrem aplikace je ClassLibrary PluginCore, obsahující abstraktní třídu Plugin, která definuje základní vlastnosti všech pluginů a třídu Utils, která je základním rozhraním pro všechny pluginy a zprostředkovává práci aplikace s jednotlivými pluginy.

Plugin Statistics obsahuje výukové moduly statistických metod, plugin Dictionary obsahuje moduly pro výuku slovníkových kompresních metod. Plugin Others, obsahuje pomocné kompresní metody a další.

6.2 Class Library PluginCore

Class Library PluginCore je jádrem aplikace, které definuje vlastnosti pluginů a zprostředkovává vyhledávání a práci s pluginy.

Utils

Abstraktní třída Utils je základním rozhraním všech pluginů. Obsahuje metodu GetPlugins, která se stará o vyhledání .dll knihoven jednotlivých pluginů.

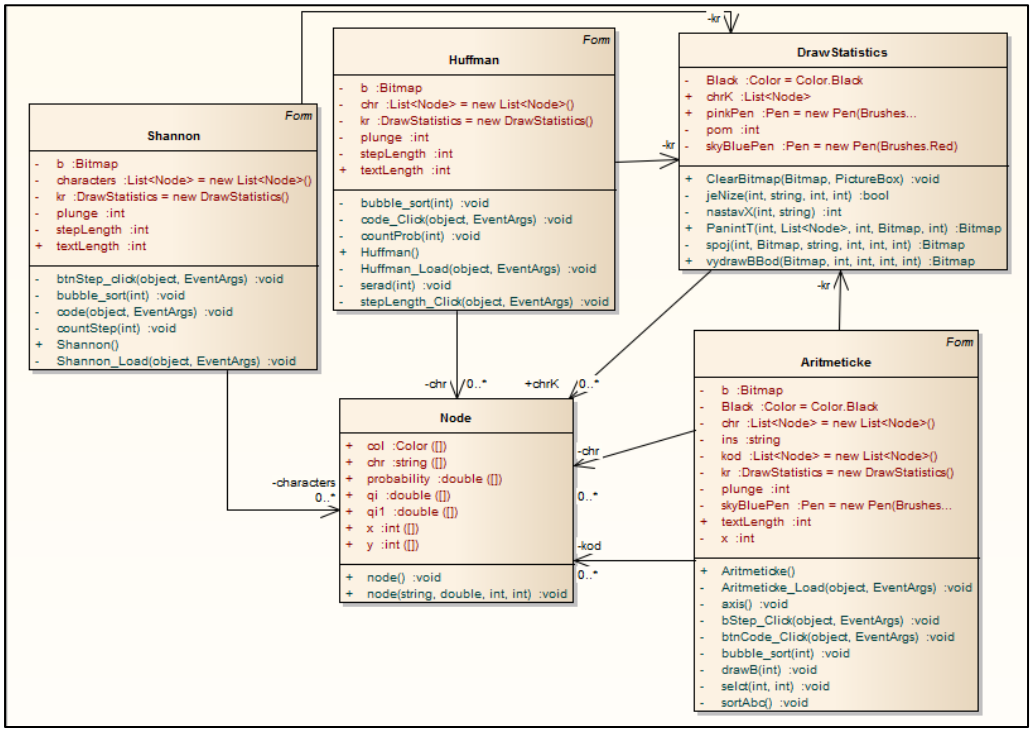
Plugin

Plugin je abstraktní třída, která definuje základní vlastnosti pluginů, což umožňuje práci s nimi bez detailnějších znalostí o jednotlivých pluginech (název a podobně). Také se stará o přiřazení vlastnosti MdiParent.

6.3 Statistics

Plugin Statistics prezentuje Statistické kompresní metody Huffman, Shannon-fan a Aritmetické kódování. Všechny tři metody využívají třídu DrawStatistics pro mazání pictureBoxu a vykreslování stromu.

Metody dále také využívají třídu Node, která definuje vlastnosti jednotlivých uzlů stromu, jako je jejich hodnota, umístění a pravděpodobnost.



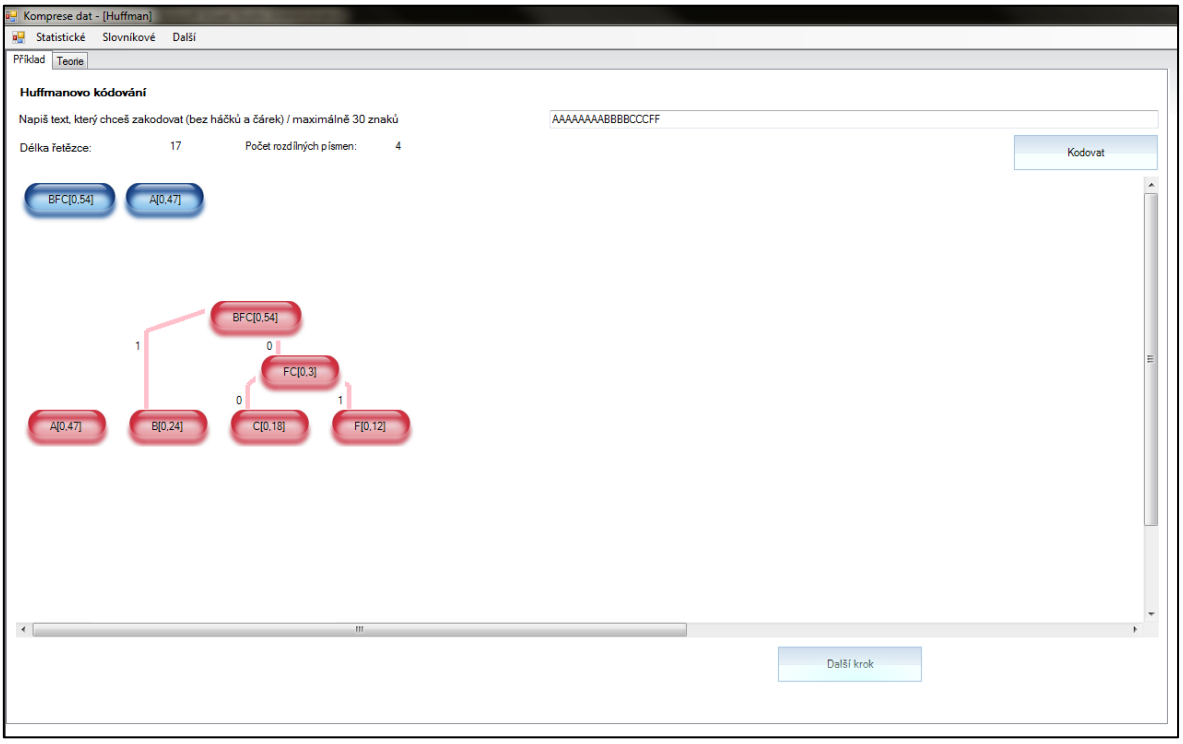
Obrázek 6.2: Class diagram Statistics

6.3.1 Draw

Třída Draw se stará o vykreslování binárních stromů a mazání picture boxu.

Metoda PaintT vykresluje binární strom vždy až do fáze (kroku), ve které se v příkladu komprese nacházíme. Nejdříve rozliší, zda se jedná o strom, jenž je třeba vykreslovat od kořene (Shannon-fano) nebo zda se vykresluje od listů (huffmanovo kódování).

U jednotlivých uzlů, které ještě nebyly v předešlých krocích vykresleny, vypočítá a zaznamená jejich pozici. Spojí jednotlivé body s jejich předchůdci a přiřadí jim binární ohodnocení. Metoda vykresluje červené uzly stromu a modré uzly souží k orientaci, ve které fázi kódovaní se nacházíme a jaké je pořadí (četnosti) jednotlivých uzlů. Což je důležité pro ohodnocení hran, jelikož by bylo zmatené měnit neustále pořadí uzlů v závislosti na tom, jak se mění pořadí jejich pravděpodobností. Například když sečteme u Huffmanova kódování pravděpodobnosti všech nejnižších uzlů, je jejich společná pravděpodobnost vyšší než pravděpodobnost znaku, který má pravděpodobnost od začátku kódování nejvyšší - například má-li znak "A" pravděpodobnost 0.48 a znak "B" pravděpodobnost 0,32 a znak "C" pravděpodobnost 0,20, je znak "A" od začátku kódování vykreslován na první pozici. Nicméně po prvním kroku, kdy dojde k spojení dvou uzlů s nejnižší pravděpodobností: "B" a "C" a sečtení jejich pravděpodobností: 0,32+0,20=0,52, je pravděpodobnost uzlu "BC" vyšší než pravděpodobnost uzlu "A". To se projeví změnou pořadí modrých uzlů v horní části formuláře, ale pořadí uzlů stromu se pro přehlednost nemění, pořadí se vyjádří pouze ohodnocením hran.



Obrázek 6.3: Binární strom

Každý uzel má pozici x a y, také svou pravděpodobnost a svůj název, kterým je znak nebo řetězec.

Uživatel má možnost vložit až 30 libovolných znaků. Po stisknutí tlačítka Kódovat se provede první krok metody a vykreslí se listy binárního stromu. Pomocí tlačítka Další krok potom uživatel může krokovat celý průběh vykreslování binárního stromu.

Po vykreslení kořene stromu aplikace uživatele upozorní, že dosáhl konce výpočtu.

6.3.2 Node

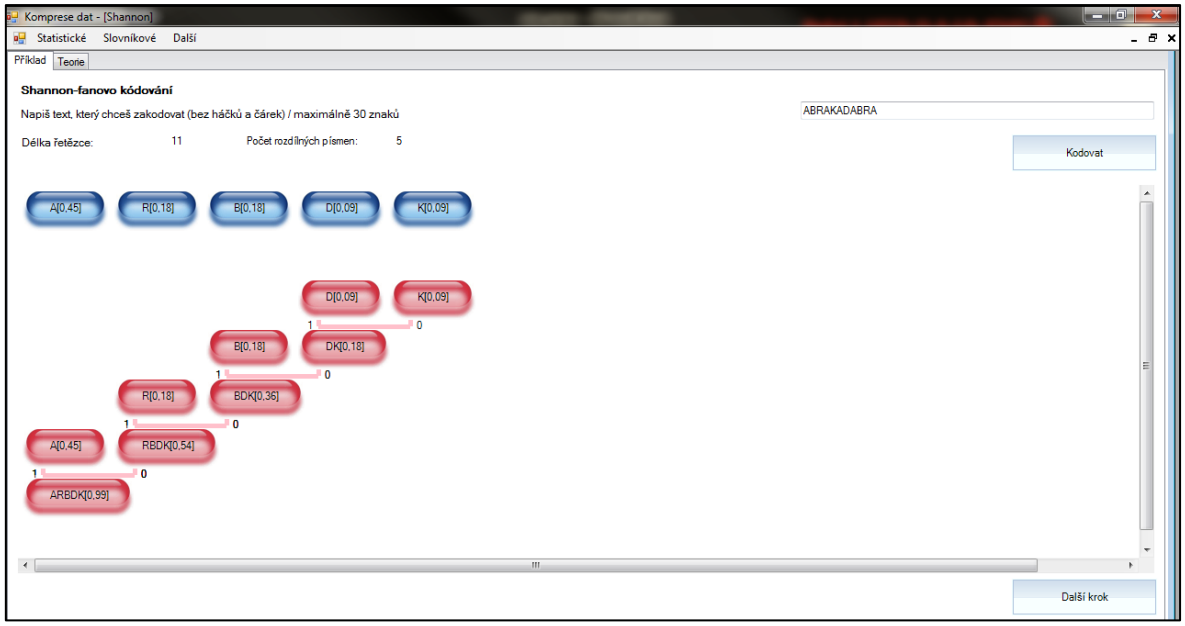
Třída Node, definuje objekt pro jednotlivé uzly binárního stromu. Umožňuje uzlu přidat vlastnost znak (řetězec), který je uzlu přiřazen, pravděpodobnost daného uzlu, pozici x a y pro vykreslení a také hodnoty q_i a q_{i-1} , které vyžívá algoritmus pro výpočet a vykreslení aritmetického kódovaní. Přičemž pozice v poli u dané vlastnosti umožňuje orientaci v aktuálním zanoření binárního stromu.

6.3.3 Shannon

Slouží k objasnění principu Shannon-fano kódování. Algoritmus v prvním průchodu počítá četnosti jednotlivých znaků. Metoda vytvoří seznam objektů typu Node, a na úrovni zanoření 0 vloží jednotlivé znaky ze zadaného řetězce. V této chvíli jsou hodnoty x a y všech uzlů rovny 0.

Vytváření nových uzlů, rozdělením pravděpodobnosti na dvě přibližně stejné poloviny provádí v každém kroku metoda CountProb*(int zan)*.

Metoda B*ubble_sort(int zan)* v každém kroku řadí uzly podle četnosti.

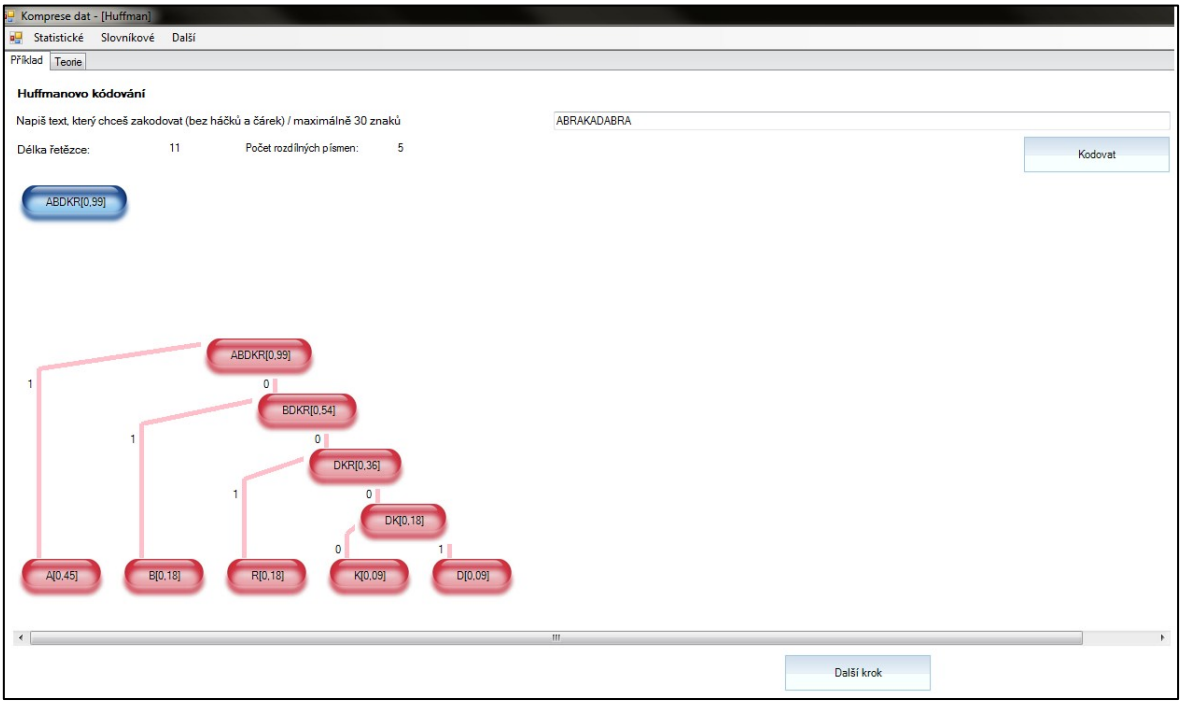


Obrázek 6.4: Implementace Shannon-fano

6.3.4 Huffman

Základní princip je prakticky stejný jako u metody pro Shannon-fano. V prvním průchodu se spočítají četnosti jednotlivých znaků, které se vloží do seznamu objektů Node v úrovni zanoření 0.

Opět při každém kroku dojde k seřazení všech uzlů v aktuálním zanoření a metoda *CountProb(int zan)* spojí uzly s nejmenší pravděpodobností.



Obrázek 6.5: Huffman

Následně se volá vykreslování binárního stromu, které počítá pozice x a y a vykresluje samotný strom.

6.3.5 Aritmetické

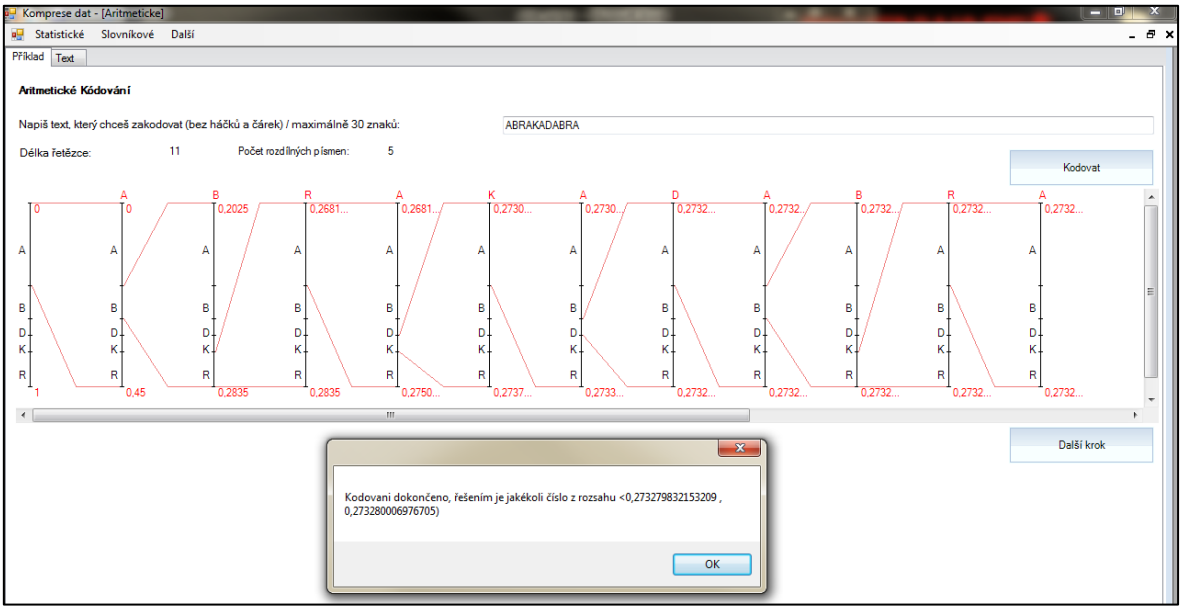
Po spuštění kódování pomocí aritmetického kódování se provede obdobné spočítání četností jako u předešlých algoritmů.

Dále obsahuje metodu *SortAbc(),* která seřadí uzly podle abecedy.

Metoda *Axis()* vykresluje v každém kroku všechny osy včetně popisů až po aktuální zanoření a počítá aktuální kód. Metoda také vyznačí intervaly jednotlivých znaků.

Metoda *Select()* vykresluje pomocné čáry, které pomáhají zdůraznit postup kódovaným řetězcem. Dále označuje, který interval je v aktuálním kroku vybrán pro výpočet intervalu pro další kroky.

Metoda *DrawB()* se stará o volání jednotlivých metod a vybírá další znak, který se má zakódovat.

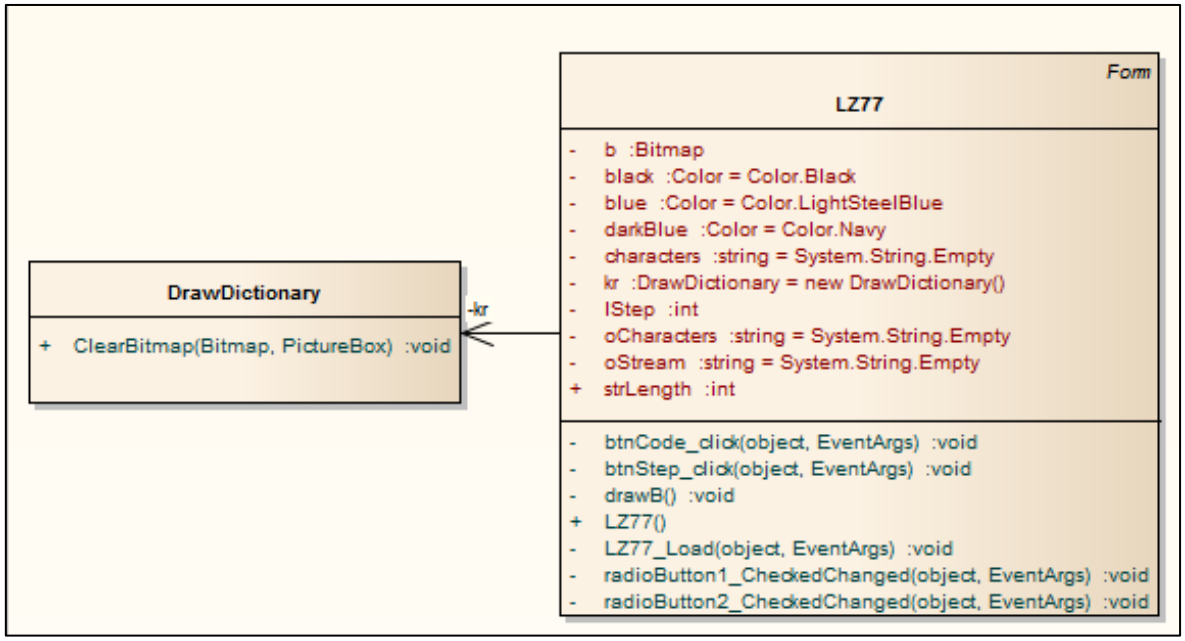


Obrázek 6.6: Aritmetické kódování

U všech metod je vypočítána také informace o počtu rozdílných znaků a celkovém počtu písmen.

6.4 Dictionary

Plugin Dictionary obsahuje algoritmus pro prezentaci metody LZ77. V budoucnu by bylo vhodné obohatit jej o metodu LZ78 a LZW, popřípadě DEFLATE.

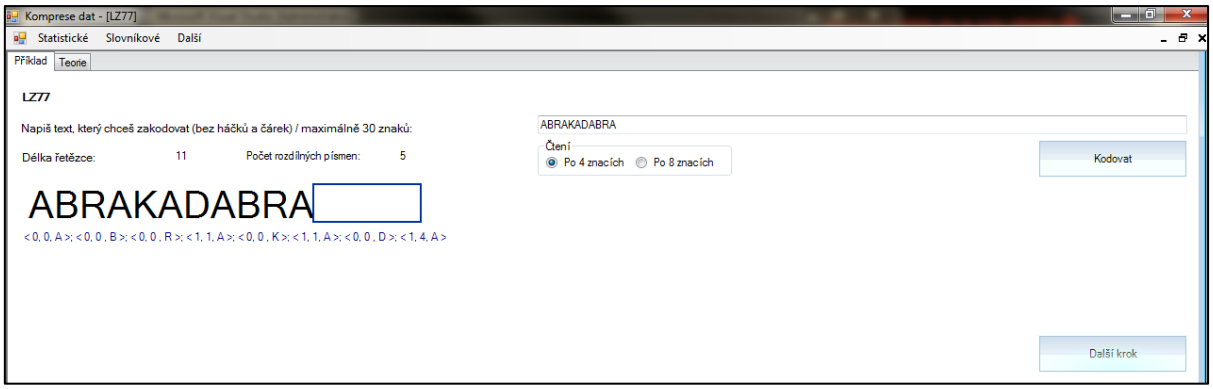


Obrázek 6.7: Class diagram Dictionary

Třída *DrawDictionary()* poskytuje metodu pro vymazání bitmapy.

6.4.1 LZ77

Obsahuje metody pro vykreslení a pro samotné kódování. Uživatel si může vybrat, zda chce použít kódování po 4 nebo po 8 znacích.

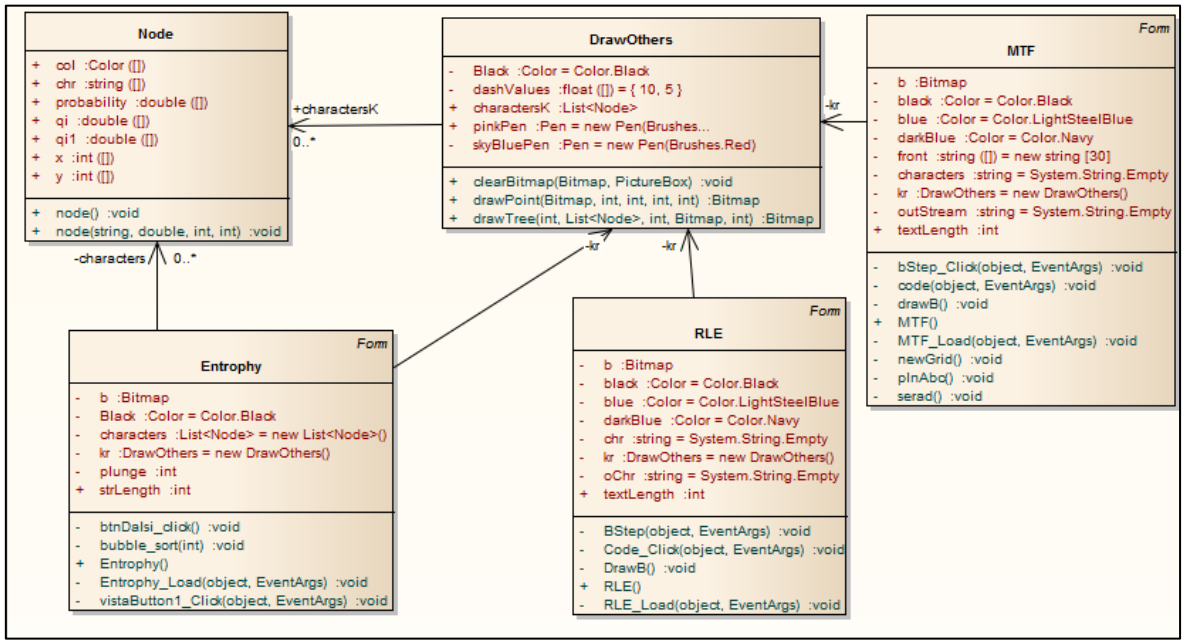


Obrázek 6.8: LZ77

V pictureBoxu je v každém kroku vykreslován zdrojový kód a také aktuální obsah výstupního souboru. Po zdrojovém souboru se posunuje posuvné okno, které označuje aktuálně kódovanou část řetězce.

6.5 Others

Plugin Others obsahuje části k vysvětlení problematiky entropie, jednoduchého kompresního algoritmu RLE a MTF.

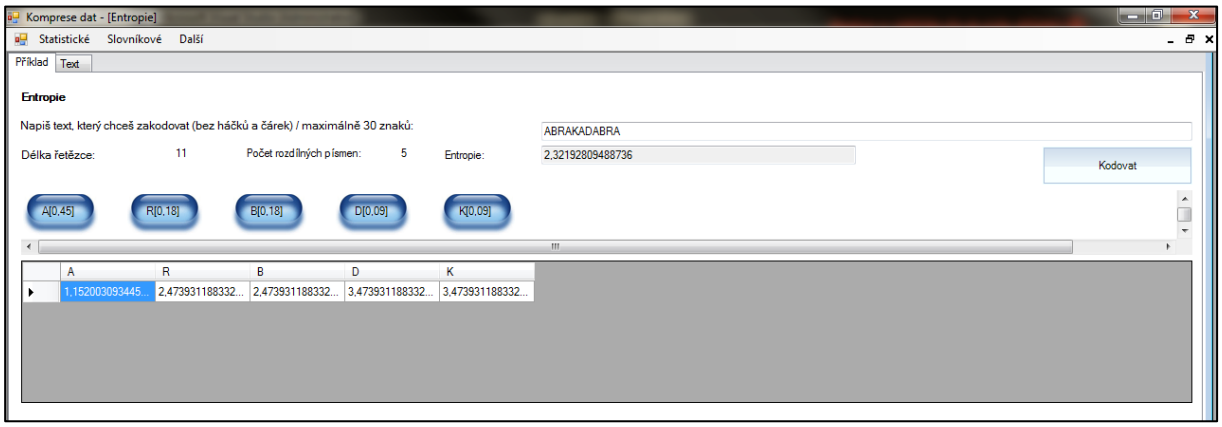


Obrázek 6.9: Class diagram Others

Používá třídu *DrawOthers* k vykreslování uzlů a mazání bitmapy, a také třídu *Node*.

6.5.1 Entropie

Třída entropie obsahuje metody pro vypočet pravděpodobností jednotlivých znaků a jejich entropie, dále také metodu pro jejich seřazení a výpočet entropie zadaného řetězce. Entropie jednotlivých znaků jsou prezentovány v tabulce.



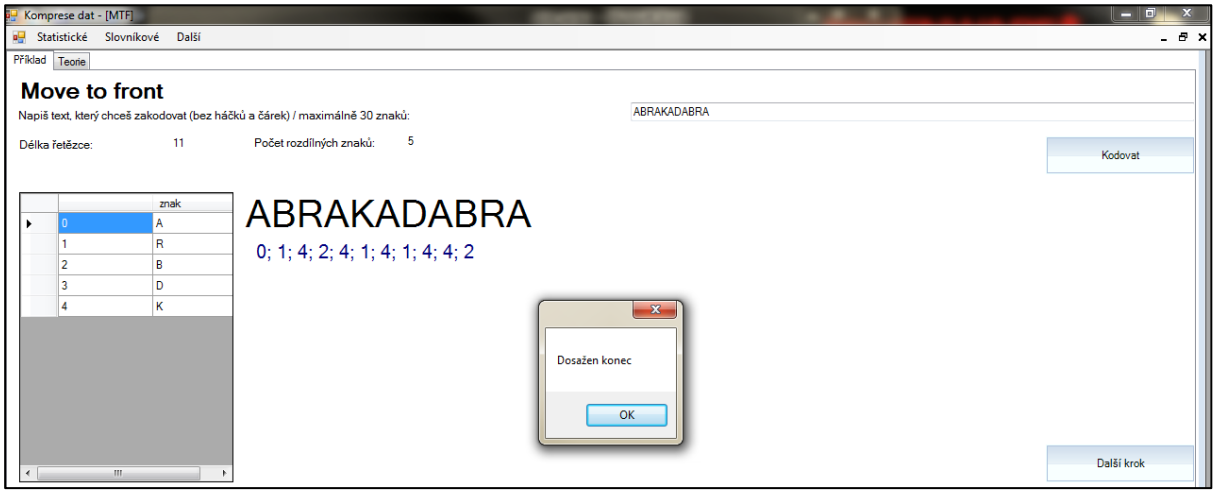
Obrázek 6.10: Entropie

Po stisku tlačítka kódovat je vypsána informace o celkovém počtu všech znaků a počtu rozdílných znaků, které uživatel vložil do textBoxu.

6.5.2 MTF

Slouží k vysvětlení problematiky pomocného algoritmu Move to Front. Obsahuje metody pro vykreslení bitmapy *DrawB()*, naplnění tabulky *NewGrid()*.

Pro přehlednost bylo rozhodnuto, že tabulka nebude obsahovat všechny znaky abecedy, ale jen ty, které se vyskytují v zdrojovém řetězci zadaném uživatelem.



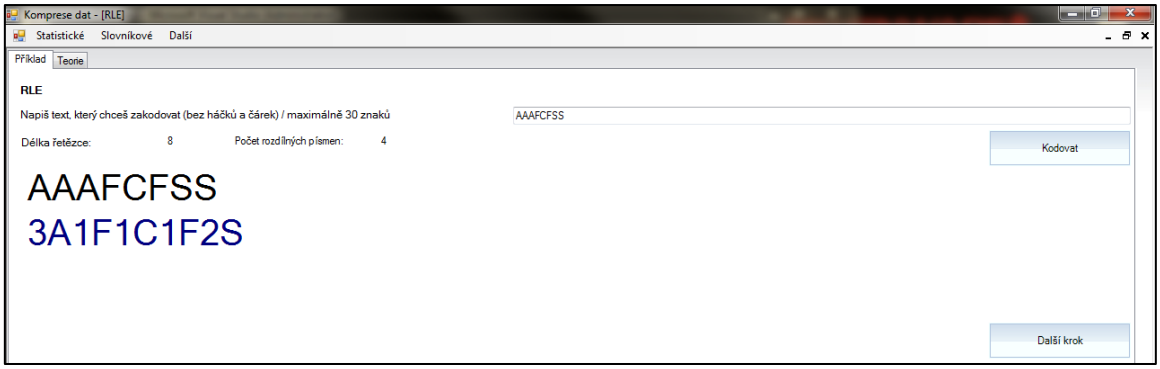
Obrázek 6.11: MTF

Uživateli se opět nabízí možnost krokování výpočtů, přičemž aktuální pořadí znaků je vypisováno do Gridu. V gridu je vypisován index jednotlivých znaků a samotné znaky.

Opět je vypisována informace o počtu znaků a počtu rozdílných znaků zadaného zdrojového souboru.

6.5.3 RLE

Slouží k vysvětlení Run Len Encoding. Obsahuje metodu pro výpočet základních statistik, metodu pro vykreslení *DrawB()*, a metodu pro samotný výpočet komprese *BStep()*.

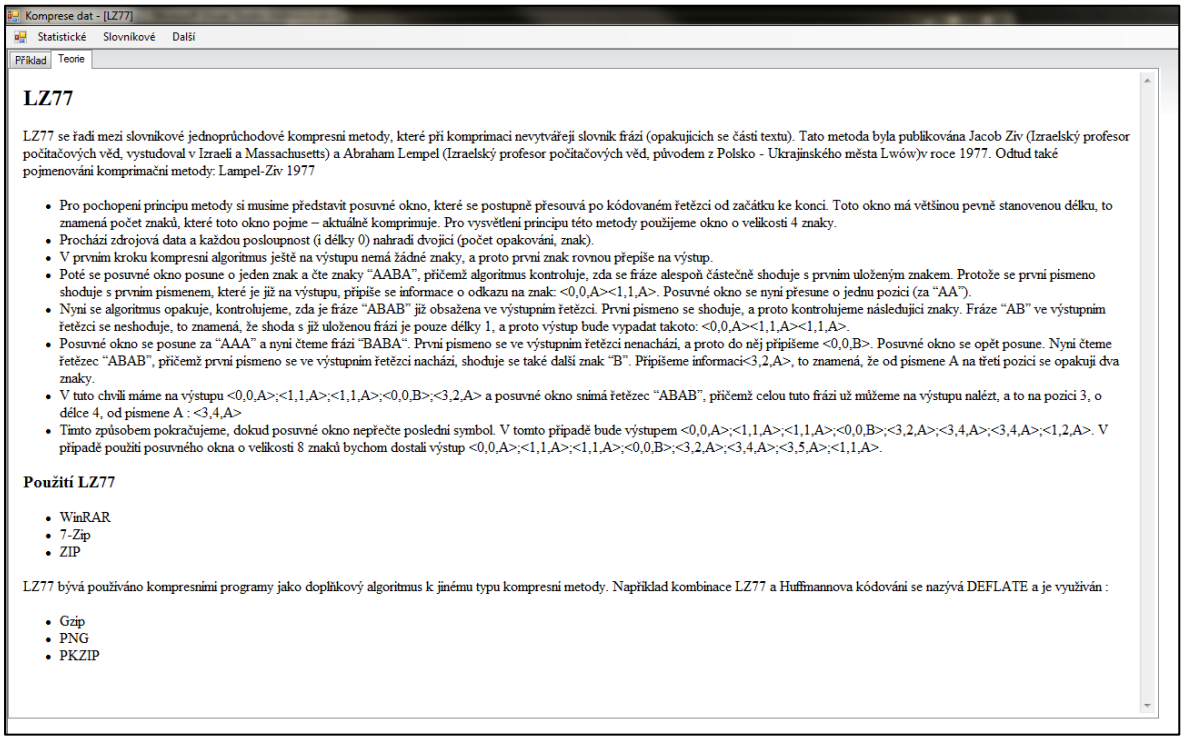


Obrázek 6.12: RLE

Uživateli se nabízí funkce krokování, v pictureBoxu je vypisován zdrojový řetězec a výstup kodéru.

6.6 Teorie

Vzhledem k účelu aplikace obsahují jednotlivé formuláře s testovacím rozhraním pro jednotlivé algoritmy také HTML stránku se základním vysvětlením teorie k dané problematice. To je umožněno pomocí komponenty TabControl, kdy na jedné záložce je vždycky testovací rozhraní a na druhé základní vysvětlení dané problematiky.



Obrázek 6.13: Teorie LZ77

Obsahem panelu teorie jsou části textu vysvětlující danou problematiku, které byly uvedeny výše.

7 Závěr

Cílem diplomové práce bylo vytvořit aplikaci pro podporu výuky předmětu Komprese dat. Za tímto účelem byla vytvořena aplikace Komprese dat obsahující pluginový systém věnující se dané problematice. Ke každému ze zpracovaných algoritmů je v aplikaci k dispozici strana s teorií vysvětlující daný problém a také testovací rozhraní, kde je možné si daný algoritmus na krátkém řetězci otestovat. Práce se snaží o objektivní osvětlení daných problematik jejich přehlednou prezentaci a testování.

V budoucnu by bylo vhodné aplikaci rozšířit o další vyučované metody, jako jsou v rámci pluginu Slovníkových metod: LZW, LZ78, DEFLATE a v rámci Statistických kompresních metod například metodu kombinující Huffmanovo kódování a Shannon-fano kódování.

Použitá literatura

1. **Večerka, Arnošt.** Komprese dat. *http://phoenix.inf.upol.cz/.* [Online] 2008. [Citace: 1. duben 2012.] http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf.

2. **Solomon, David.** *Data Compression.* New York : Springer, 2004. ISBN 0-387-40697-2.

3. **Pu, Ida Mengyi.** *Fundamental Data Compression.* Oxford : Elsevier, 2006. ISBN-10 0-7506-6310-3.

4. **Morkes, David.** *Komprimační a archivační programy.* Praha : Computer Press, 1998. ISBN 80-7226-089-8.

Systém pro podporu výuky v předmětu komprese dat

Spustitelný systém je součástí CD přiloženého k této práci:

DataCompression/bin/Crossroad.exe